

Cycle - Solution

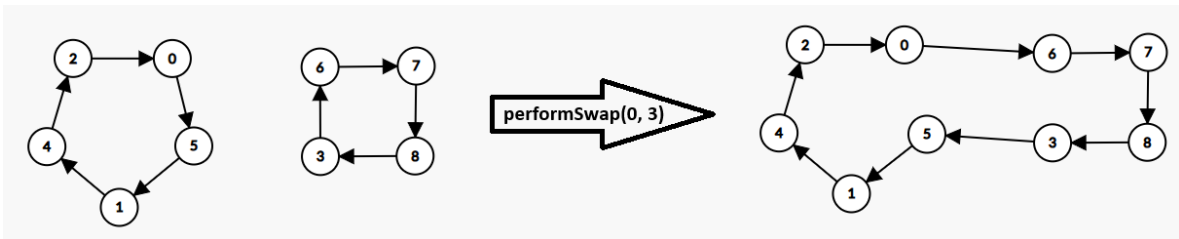
Atanas Dimitrov, Encho Mishinev

Core observation

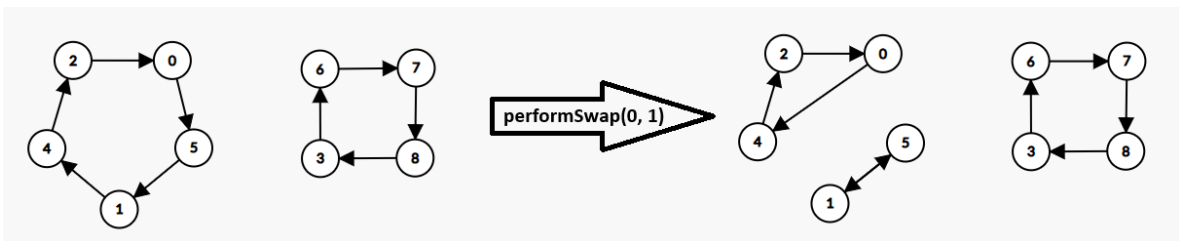
To be able to reason about anything in this problem, we must be able to interpret how the number of cycles changes after a swap. Let's visually look at one permutation to get a sense of this. A standard way to visualize cycles of a permutation is for nodes to correspond to positions, with node i having a single outgoing edge to node P_i .

Let's consider the permutation $[5, 4, 0, 6, 2, 1, 7, 8, 3]$ and the swaps $(0, 3)$ and $(0, 1)$.

Swapping two elements that are in different cycles merges them:



Swapping two elements that are in the same cycle splits it into two:



This is going to be a fundamental idea for the solution. The answer to a given query will always be 1 more or 1 less compared to the previous one, depending on whether the pair we swap is in the same cycle or not.

Sorting the permutation is equivalent to making exactly N cycles.

Subtask 1

We can get the initial amount of cycles by doing two swaps of $(0, 1)$. Then for each even i , we can try to swap (P_i, P_{i+1}) . If the number of cycles increases, then we have changed $(i + 1, i)$ to $(i, i + 1)$ and we can continue to the next i . Otherwise if the number of cycles decreases we should swap (P_i, P_{i+1}) again to revert to the original $(i, i + 1)$.

This takes about $2N$ swaps at worst, which easily gets full score.

Subtask 2

Since there is only a single swap that is necessary in the beginning permutation, it may be intuitive to try various swaps and then revert them. However, it is not hard to see that if we always revert our action, we cannot find the right pair in less than $O(N^2)$ queries.

The restriction implies that the initial amount of cycles is exactly $N - 1$. Let the single swap needed to sort the permutation be (A, B) (with $A < B$). Let us try the swaps $(0, 1), (0, 2), \dots, (0, N - 1)$. Since any element i such that $i \neq A$ and $i \neq B$ is a single-node cycle, executing $(0, i)$ will merge it with 0's cycle and decrease the number of cycles. When $(0, A)$ is executed it will also decrease the number of cycles. However, the $(0, B)$ swap will be the first to increase the number of cycles because at that moment it is guaranteed that 0 and B are in the same cycle, since 0 and A are.

Thus we get a way to find B in at most N operations. After that we can revert all swaps we've done so that we are in the initial position. Once we know B we can just try each possible A – if the swap results in N cycles we're done, otherwise we revert and try another A . In total we do about N operations to find B , another N for reverting to the starting state, then at most $2N$ for finding A , for a total of $4N$. This yields full score for the subtask.

Subtask 3

To solve the full problem we must realize that reaching the state of having just a single cycle is quite powerful. We can use a very similar procedure as the one in subtask 2 to obtain a single cycle. We try swaps $(0, 1), (0, 2), \dots, (0, N - 1)$. If a swap decreases the number of cycles, we continue onwards. If a swap increases the number of cycles, we revert it. Visualizing this procedure, it is easy to see that we are simply merging all cycles into the cycle of node 0. There are many other ways we could achieve this, some on average using less swaps, but this approach is sufficient. This part takes at most $2N$ operations, since we may have to revert almost every swap.

Now consider having a permutation that has a single cycle. Take some pair A, B and perform the swap (A, B) . It is guaranteed that this will split the permutation into two cycles, one containing A and one containing B . Now for every other node i , we can try swapping (A, i) and then revert that swap. If the swap increases the number of cycles then i was in A 's cycle, otherwise it was in B 's cycle. Using this procedure we can split the single cycle into two subcycles and find the exact set of elements in each, taking at most $1 + 2(N - 2) = 2N - 3$ operations. Afterwards those cycles can be solved fully independently.

At this point the parallel with quicksort should be obvious. We have a linear way of splitting the elements in two independent groups, and we can solve those groups recursively. Thus it is natural to just pick A and B at random to get a total of $O(N \log N)$ swaps. The exact expected constant can be proven to be about $2.8N \log N$. This can be shown by any standard method for proving the average case complexity of quicksort and multiplying by 2 (since we need $2N$ operations instead of N for each level). There is a very small chance that a very unlucky contestant might exceed the 30000 swap limit, but one can always resubmit with a different random seed.